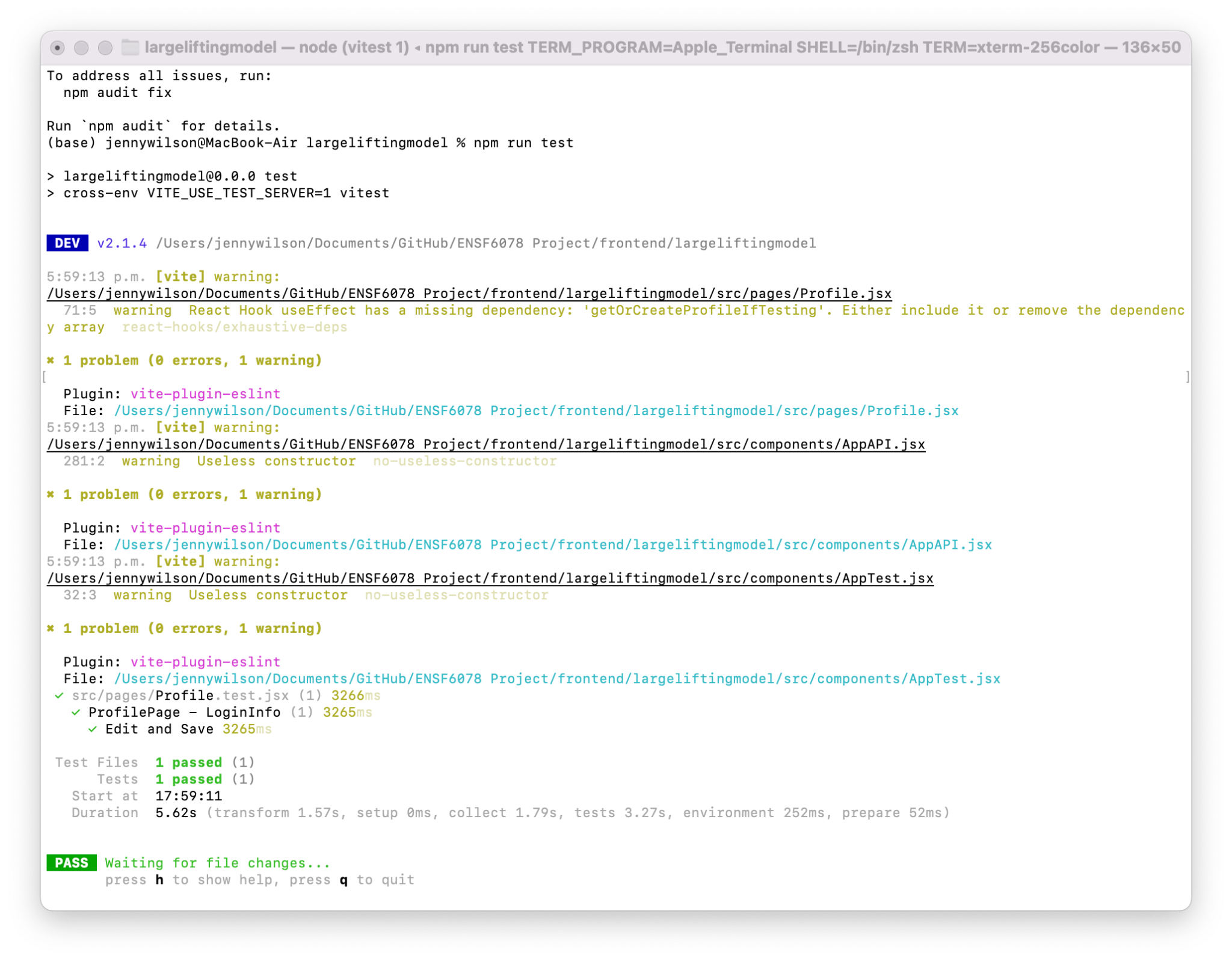
**Testing Document**

**Group 1**

**Frontend:**

Exploratory testing was the primary form of testing the frontend of our application. This included: verifying web pages rendered correctly, web pages are responsive for various devices, ensuring modules and components worked as expected, confirming links direct users to the correct pages, etc. In addition to the exploratory testing, some unit tests were performed to validate our key react components. The method of validation was largely focussed on observing and interacting with our application during exploratory testing. Issues encountered during exploratory testing were documented, and addressed.

Example- Unit Test for Profile page



**Backend:**

For the backend we used a combination of system tests and unit tests to confirm the functionality. The unit tests were used to validate our backend components worked correctly with isolation from the dependencies the components needed (mock data used instead of using the dependencies). The system tests were used to validate the interactivity of our components, confirming that each component works properly together. Additionally we tested our API calls using Postman, allowing for the validation of our endpoints.

Validation Plan-

User test code-

from django.urls import reverse

from rest\_framework import status

from rest\_framework.test import APITestCase

from django.contrib.auth import get\_user\_model

from rest\_framework\_simplejwt.tokens import RefreshToken

from unittest.mock import Mock, patch

from users.models import UserProfile

from rest\_framework.response import Response

User = get\_user\_model()

class UserGoogleLoginTests(APITestCase):

@patch('users.views.GoogleLoginView.get\_google\_user\_info')

@patch('dj\_rest\_auth.registration.views.SocialLoginView.post')

def test\_google\_login\_new\_user(self, mock\_super\_post, mock\_get\_google\_user\_info):

# Mock the response from the Google API

mock\_get\_google\_user\_info.return\_value = {

'email': 'testuser@example.com',

'given\_name': 'Test',

'family\_name': 'User'

}

# Simulate the response from super().post()

mock\_super\_post.return\_value = Response(status=status.HTTP\_200\_OK)

# Now make a request to the Google login endpoint

url = reverse('auth\_social\_google')

data = {'access\_token': 'fake-access-token'}

response = self.client.post(url, data, format='json')

# Assert that the response is successful

self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

# Check that the user was created

user = User.objects.get(email='testuser@example.com')

self.assertIsNotNone(user)

self.assertEqual(user.first\_name, 'Test')

self.assertEqual(user.last\_name, 'User')

# Check that tokens are returned in the response

self.assertIn('access', response.data)

self.assertIn('refresh', response.data)

@patch('users.views.GoogleLoginView.get\_google\_user\_info')

@patch('dj\_rest\_auth.registration.views.SocialLoginView.post')

def test\_google\_login\_existing\_user(self, mock\_super\_post, mock\_get\_google\_user\_info):

# Create a user that already exists

existing\_user = User.objects.create\_user(

username='testuser',

email='testuser@example.com',

first\_name='Existing',

last\_name='User'

)

# Mock the response from the Google API

mock\_get\_google\_user\_info.return\_value = {

'email': 'testuser@example.com',

'given\_name': 'Existing',

'family\_name': 'User'

}

# Simulate the response from super().post()

mock\_super\_post.return\_value = Response(status=status.HTTP\_200\_OK)

# Now make a request to the Google login endpoint

url = reverse('auth\_social\_google')

data = {'access\_token': 'fake-access-token'}

response = self.client.post(url, data, format='json')

# Assert that the response is successful

self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

# Check that the user still exists and was not duplicated

user\_count = User.objects.filter(email='testuser@example.com').count()

self.assertEqual(user\_count, 1)

# Check that tokens are returned in the response

self.assertIn('access', response.data)

self.assertIn('refresh', response.data)

class UserProfileTests(APITestCase):

def setUp(self):

# Create a test user and retrieve tokens

self.user = User.objects.create\_user(

username='testuser',

email='testuser@example.com',

password='testpassword',

first\_name='Test',

last\_name='User'

)

self.refresh\_token = RefreshToken.for\_user(self.user)

self.access\_token = str(self.refresh\_token.access\_token)

def test\_get\_user\_profile(self):

# Authorize with JWT token

self.client.credentials(HTTP\_AUTHORIZATION='Bearer ' + self.access\_token)

url = reverse('user\_profile')

response = self.client.get(url)

# Assert profile data is returned successfully

self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

self.assertIn('email', response.data)

self.assertEqual(response.data['email'], self.user.email)

def test\_update\_user\_profile(self):

self.client.credentials(HTTP\_AUTHORIZATION='Bearer ' + self.access\_token)

url = reverse('user\_profile')

data = {

'first\_name': 'Updated',

'last\_name': 'User'

}

response = self.client.put(url, data, format='json')

# Assert profile update is successful

self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

self.assertEqual(response.data['first\_name'], 'Updated')

self.assertEqual(response.data['last\_name'], 'User')

def test\_delete\_user\_profile(self):

self.client.credentials(HTTP\_AUTHORIZATION='Bearer ' + self.access\_token)

url = reverse('user\_profile')

response = self.client.delete(url)

# Assert deletion is successful

self.assertEqual(response.status\_code, status.HTTP\_204\_NO\_CONTENT)

self.assertFalse(User.objects.filter(id=self.user.id).exists())

class UserLogoutTests(APITestCase):

def setUp(self):

# Create a test user and retrieve tokens

self.user = User.objects.create\_user(

username='testuser',

email='testuser@example.com',

password='testpassword'

)

self.refresh\_token = RefreshToken.for\_user(self.user)

self.access\_token = str(self.refresh\_token.access\_token)

def test\_logout(self):

self.client.credentials(HTTP\_AUTHORIZATION='Bearer ' + self.access\_token)

url = reverse('auth\_logout')

data = {'refresh': str(self.refresh\_token)}

response = self.client.post(url, data, format='json')

# Assert logout is successful and token is blacklisted

self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

self.assertIn('Successfully logged out', response.data['detail'])

Workout test code-

from rest\_framework.test import APITestCase

from django.urls import reverse

from rest\_framework import status

from django.contrib.auth import get\_user\_model

from rest\_framework\_simplejwt.tokens import RefreshToken

from users.models import HealthData

from users.models import UserProfile

from llm.views import LlmPromptView

from unittest.mock import patch

User = get\_user\_model()

class CreatePromptTest(APITestCase):

def setUp(self):

# Create a user and associated UserProfile

self.user = User.objects.create\_user(

username='testuser',

email='testuser@example.com',

password='testpassword'

)

profile, \_ = UserProfile.objects.get\_or\_create(user=self.user)

# Use update\_or\_create, not create, to avoid IntegrityError!!

self.health\_data, created = HealthData.objects.update\_or\_create(

profile=profile,

defaults={

'dob': '1990-01-01',

'gender': 'Male',

'height': 1.75,

'weight': 75,

'favourite\_workout\_type': 'Cardio',

'workout\_experience': 'Intermediate',

'fitness\_goal': 'Build muscle',

'injuries': 'None',

'other\_considerations': 'None'

}

)

# Confirm data has saved correctly

self.view = LlmPromptView()

def test\_create\_prompt(self):

# Simulated data

ser\_obj\_data = {

'length': 60,

'difficulty': 'Easy',

'workout\_type': 'Weights',

'target\_area': 'Chest',

'equipment\_access': 'Full Gym'

}

prompt\_text = self.view.createPrompt(ser\_obj\_data)

# print("[TEST] Final prompt text:\n", prompt\_text)

self.assertIn("length: 60", prompt\_text)

self.assertIn("difficulty: Easy", prompt\_text)

self.assertIn("workout\_type: Weights", prompt\_text)

self.assertIn("target\_area: Chest", prompt\_text)

self.assertIn("equipment\_access: Full Gym", prompt\_text)

class CreateWorkoutTest(APITestCase):

@classmethod

def setUpTestData(cls):

cls.user = User.objects.create\_user(

username='testuser',

email='testuser@example.com',

password='testpassword',

first\_name='Test',

last\_name='User'

)

profile, \_ = UserProfile.objects.get\_or\_create(user=cls.user)

cls.health\_data, created = HealthData.objects.update\_or\_create(

profile=profile,

defaults={

'dob': '1990-01-01',

'gender': 'Male',

'height': 1.75,

'weight': 75,

'favourite\_workout\_type': 'Cardio',

'workout\_experience': 'Intermediate',

'fitness\_goal': 'Build muscle',

'injuries': 'None',

'other\_considerations': 'None'

}

)

cls.refresh\_token = RefreshToken.for\_user(cls.user)

cls.access\_token = str(cls.refresh\_token.access\_token)

@patch('workout.views.LlmConnection.requestWorkout')

def test\_post\_workout(self, mock\_requestWorkout):

# Mock the LLM response

mock\_requestWorkout.return\_value = 'Sample LLM Response'

# Verify healthdata in the test

health\_data = HealthData.objects.filter(profile\_\_user=self.user).first()

if health\_data:

# print("[DEBUG] Retrieved HealthData in test\_post\_workout:")

for field in HealthData.\_meta.fields:

name = field.name

val = getattr(health\_data, name, None)

# print(f"[DEBUG] {name}: {val}")

else:

print("[ERROR] HealthData not found in test\_post\_workout")

# Send request

self.client.credentials(HTTP\_AUTHORIZATION='Bearer ' + self.access\_token)

url = reverse('create-workout')

data = {

'length': 60,

'difficulty': 'Easy',

'workout\_type': 'Weights',

'target\_area': 'Chest',

'equipment\_access': 'Full Gym'

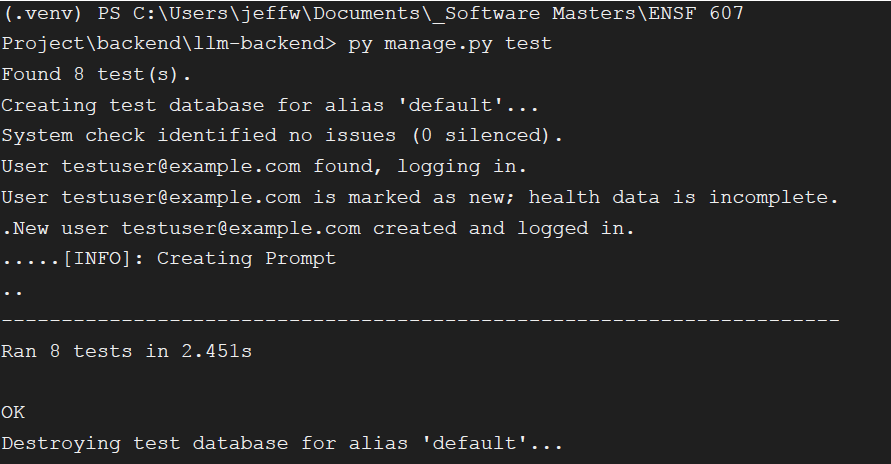
}

response = self.client.post(url, data, format='json')

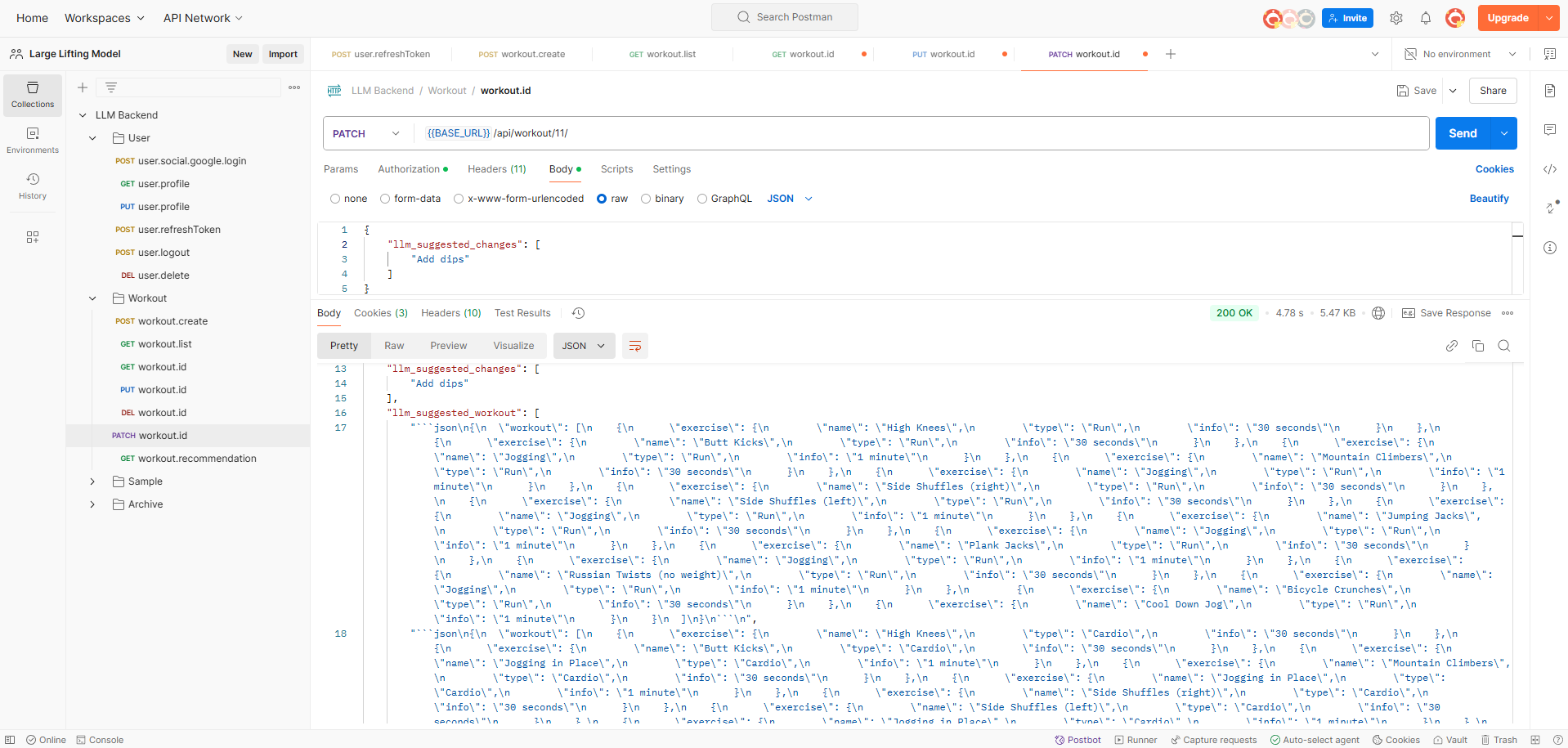
self.assertEqual(response.status\_code, status.HTTP\_201\_CREATED)

mock\_requestWorkout.assert\_called\_once()

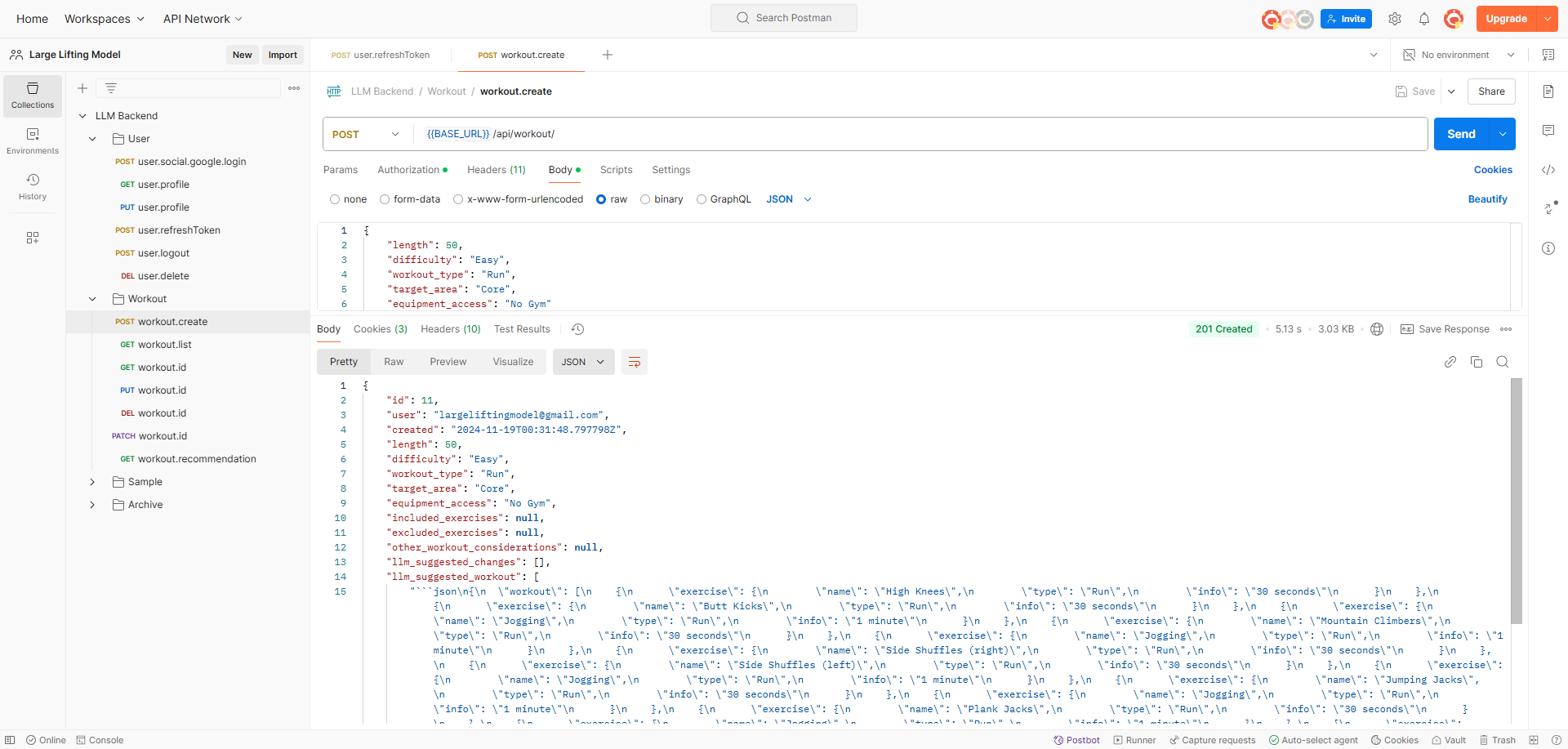
Django Tests Output-

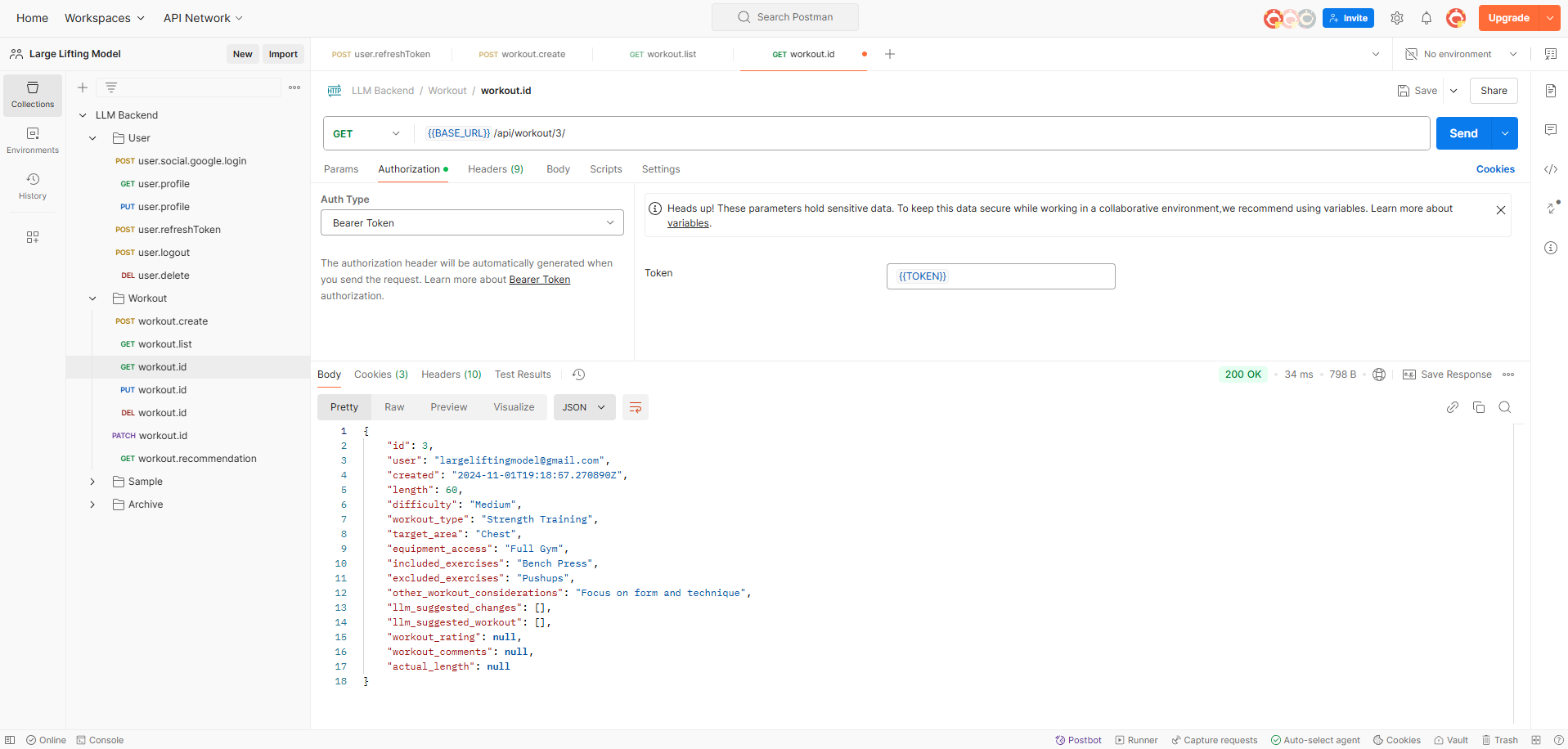


Patch Workout Test-

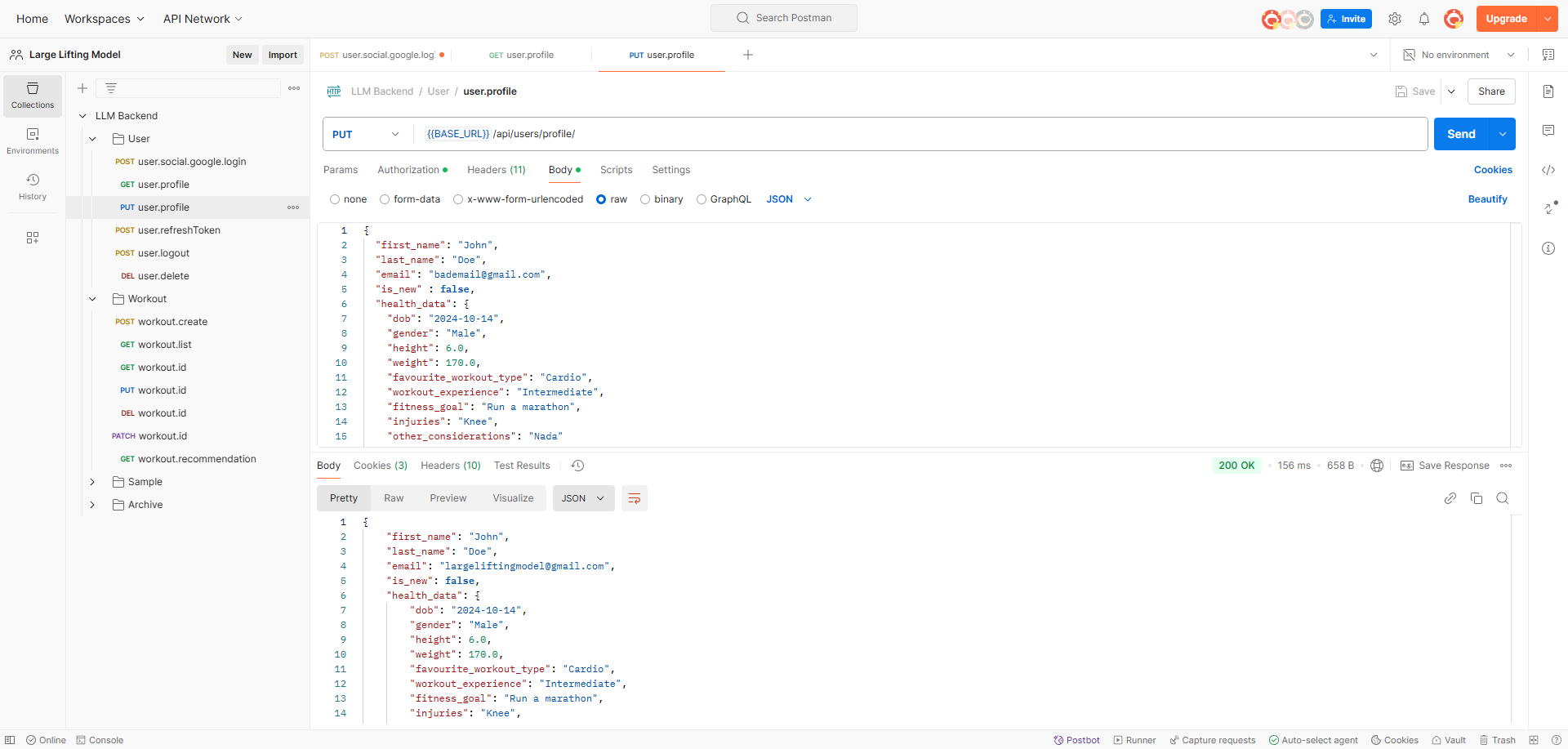


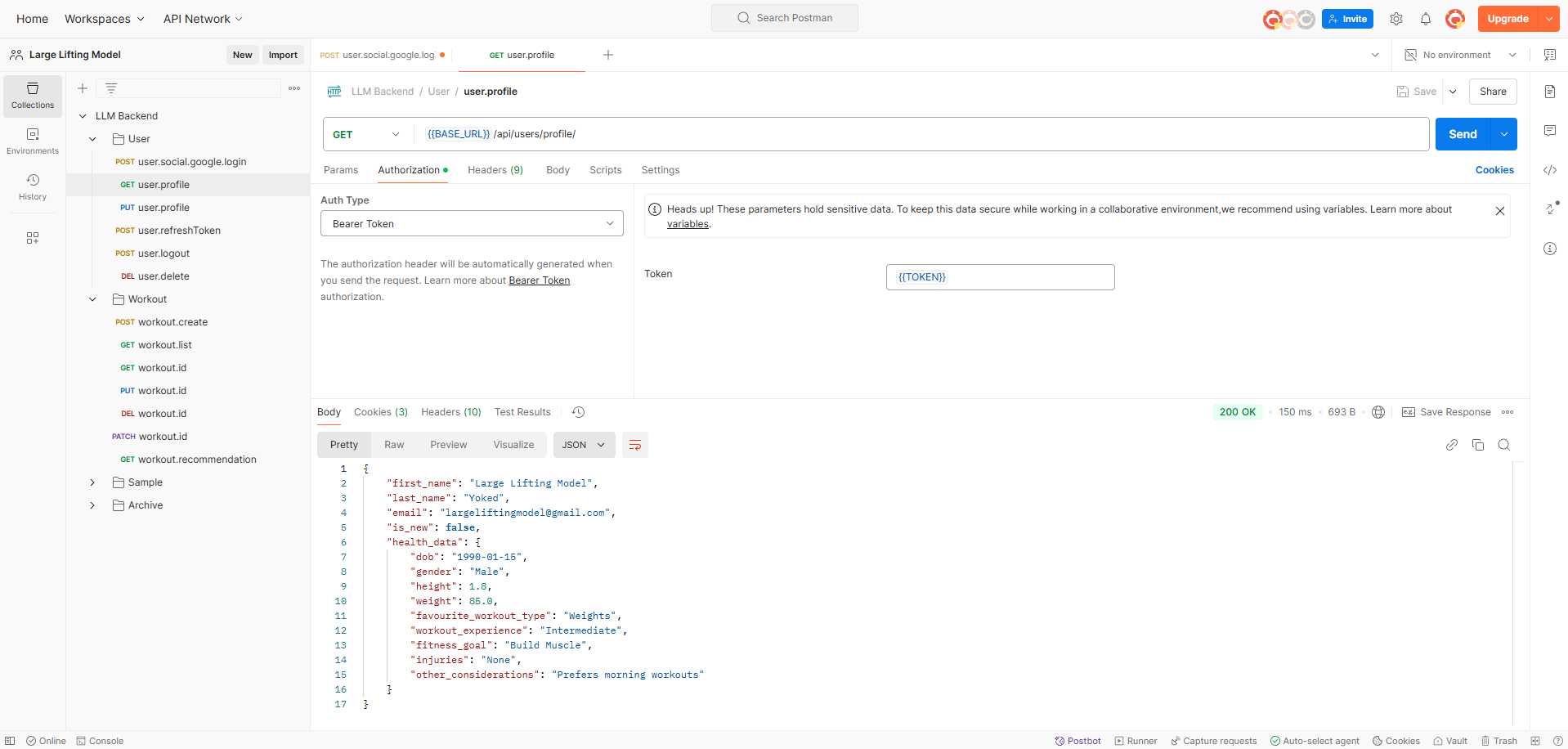
Create Workout Test-

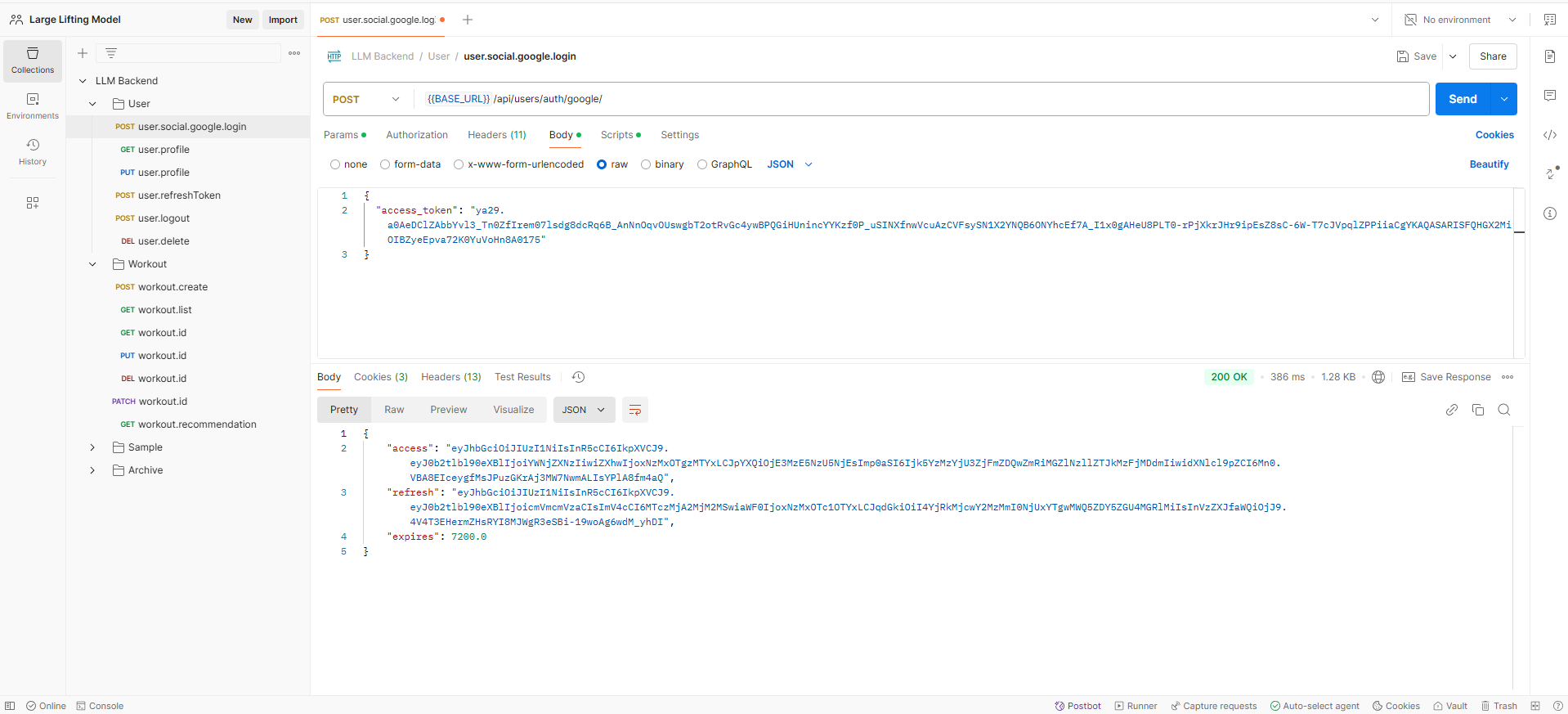


Get Workout Test- 

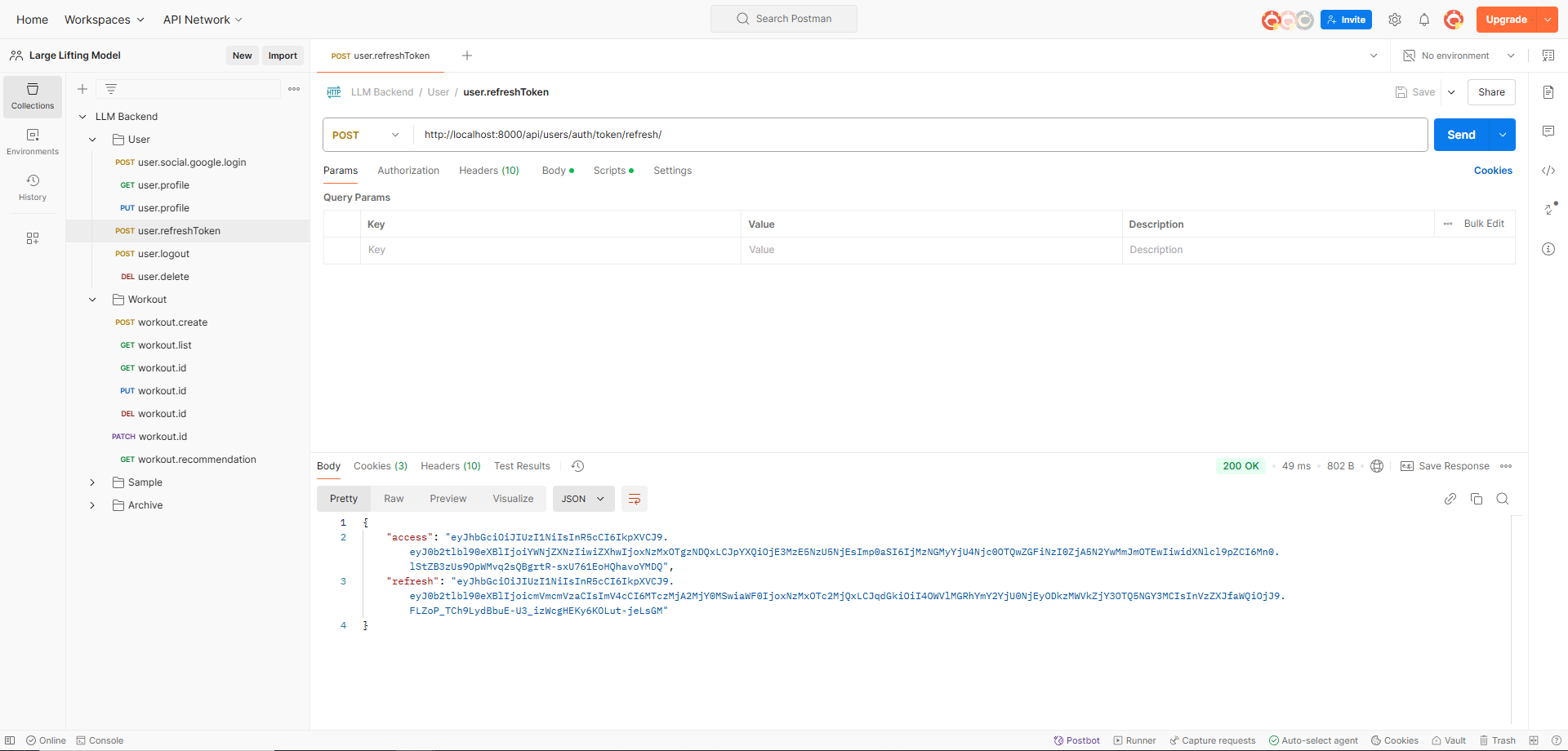
Put User Profile Test-



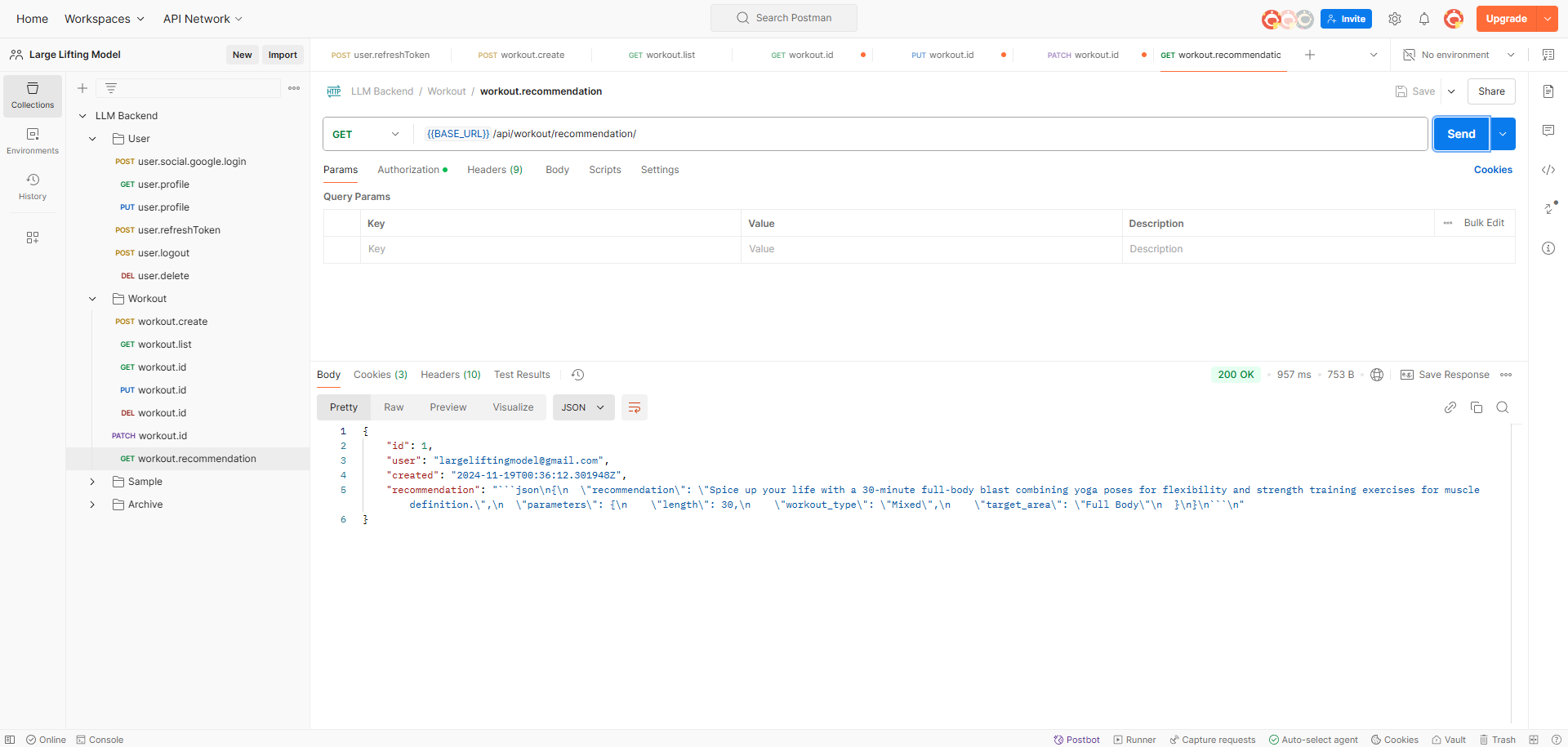
Get User Profile Test- 

Create User Google Login Test- 

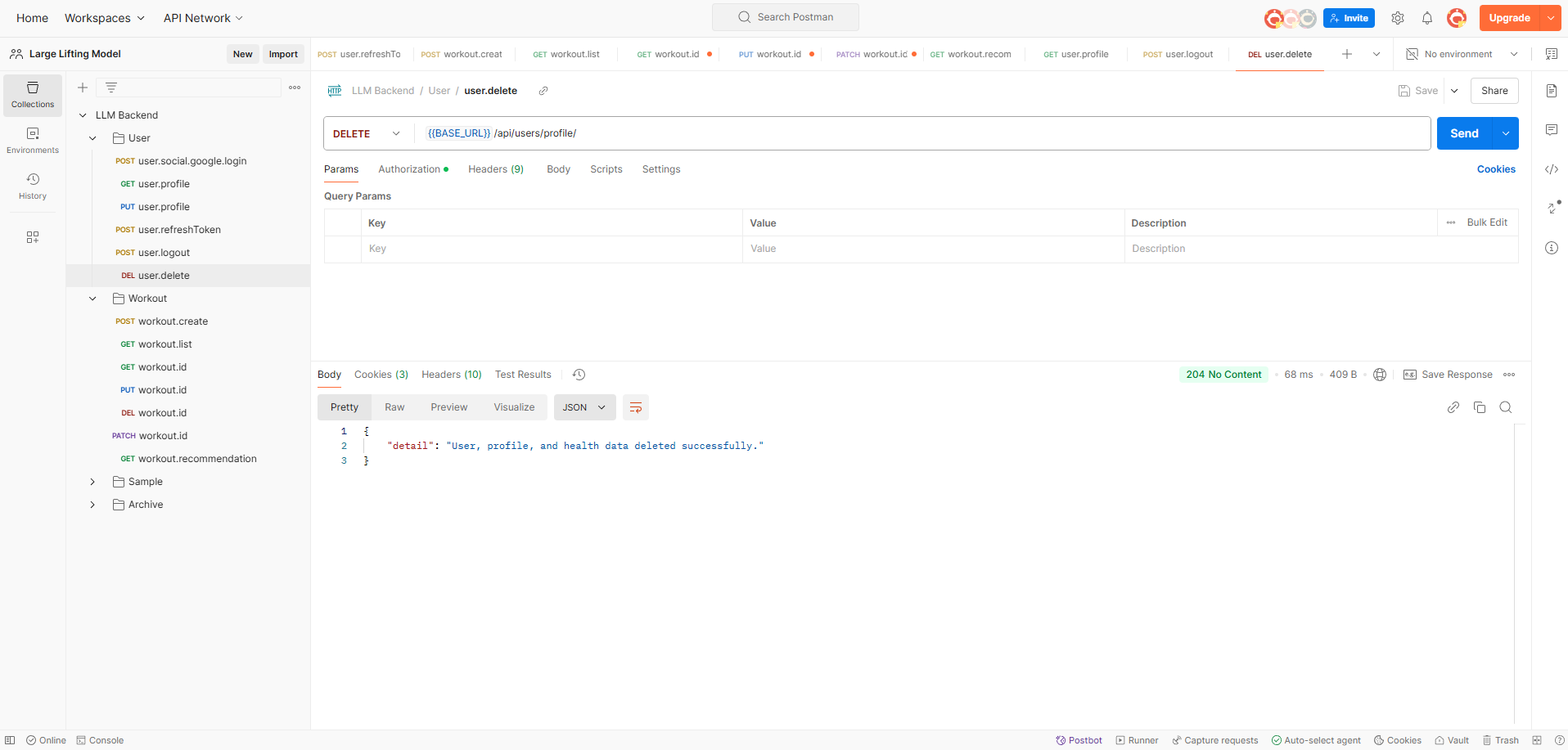
Post User Refresh Token Test-



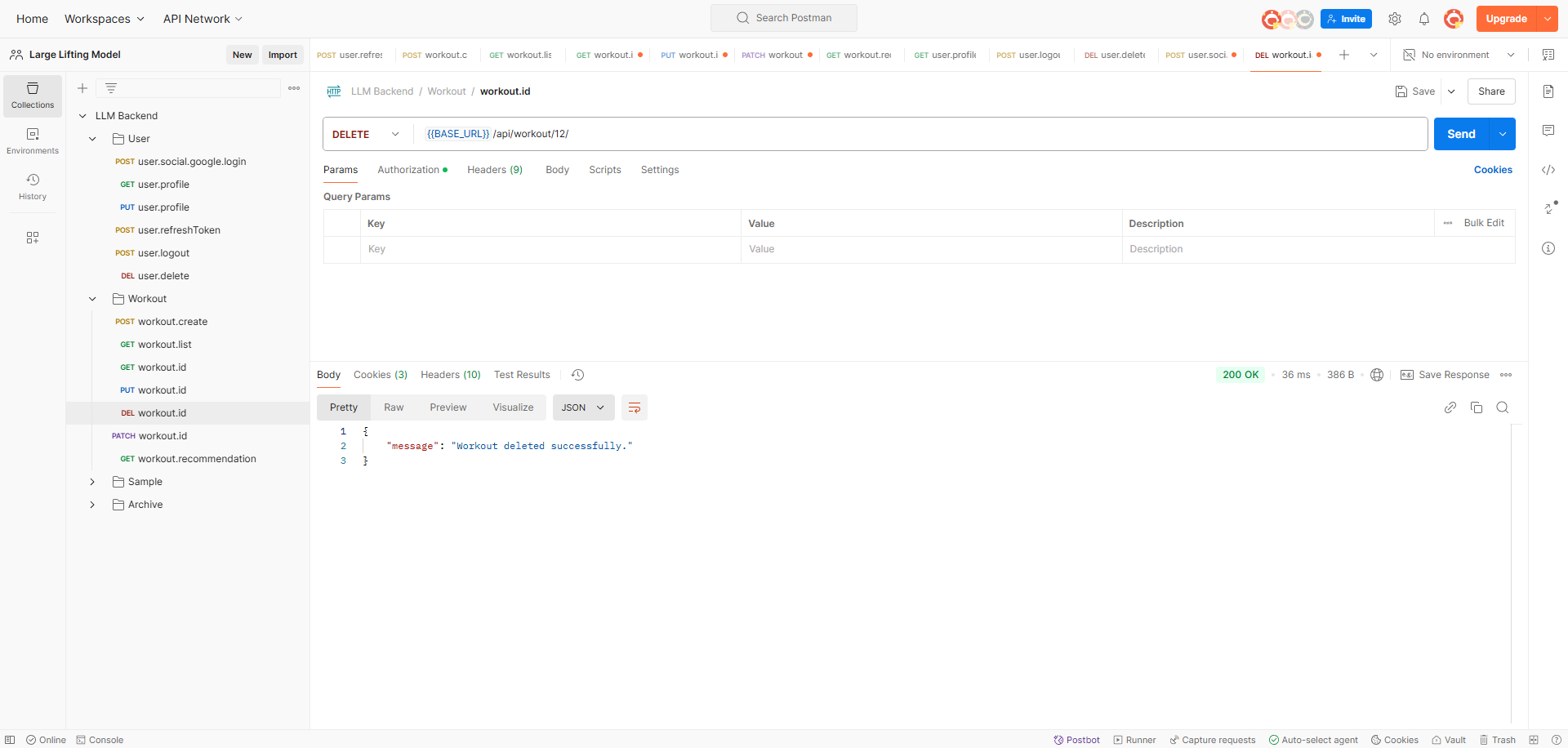
Get Workout Recommendation Test-



Delete User Test-



Delete Workout Test-



Logout User Test-

